Thuật toán Piecemeal-DFS Cho Việc Tìm Kiếm Duyệt Sâu Với Mức Năng Lượng Ràng Buộc Trên Bài Toán Hợp Hai Cây Cho Trước

Conference Paper · June 2025

CITATIONS
0

READS
5

3 authors:

Son Hong Nguyen
Posts and Telecommunications Institute of Technology - Ho Chi Minh City
19 PUBLICATIONS **59** CITATIONS
SEE PROFILE

Van Van Chau
Posts and Telecommunications Institute of Technology
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Nguyen Minh Tuan
Posts and Telecommunications Institute of Technology - Ho Chi Minh City
6 PUBLICATIONS **1** CITATION
SEE PROFILE

Thuật toán Piecemeal-DFS Cho Việc Tìm Kiếm Duyệt Sâu Với Mức Năng Lượng Ràng Buộc Trên Bài Toán Hợp Hai Cây Cho Trước

Nguyễn Minh Tuấn[1,*], Nguyễn Hồng Sơn [1,*] và Châu Văn Vân [1,*]

[1]Học viện Công nghệ Bưu chính Viễn thông cơ sở tại TP. Hồ Chí Minh
11 Nguyễn Đình Chiểu, P. Đa Kao, Q. 1, TP. Hồ Chí Minh
** Email: minhtuan@ptit.edu.vn; sonngh@ptit.edu.vn; vancv@ptit.edu.vn*

Tóm tắt:

Depth-First Search (DFS) là một trong những thuật toán cơ bản nhất để duyệt hoặc tìm kiếm trên đồ thị. DFS thường được triển khai bằng đệ quy hoặc sử dụng ngăn xếp (stack). Trong quá trình nghiên cứu và ứng dụng, DFS đã được mở rộng thành nhiều phiên bản để đáp ứng các yêu cầu khác nhau, nổi bật như: Iterative Deepening DFS (IDDFS) để kết hợp ưu điểm của DFS và Breadth-First Search (BFS), IDDFS thực hiện DFS với giới hạn độ sâu tăng dần. Phiên bản này tiết kiệm bộ nhớ như DFS nhưng tìm kiếm theo lớp như BFS; Depth-Limited DFS (DLDFS) là một phiên bản của DFS với giới hạn độ sâu tối đa. Khi đạt độ sâu giới hạn, thuật toán dừng việc đi sâu hơn; Bidirectional DFS để duyệt đồ thị từ hai hướng đồng thời: từ đỉnh bắt đầu và đỉnh kết thúc. Khi hai luồng gặp nhau, quá trình tìm kiếm hoàn tất; Randomized DFS để chọn ngẫu nhiên thứ tự các đỉnh kề để duyệt nhằm tăng tính đa dạng cho các bài toán như sinh mê cung, sinh cây ngẫu nhiên; Parallel/Distributed DFS để thực hiện đồng thời trên nhiều tác nhân hoặc máy tính khác nhau nhằm tăng tốc độ khám phá đồ thị rất lớn. Trong bài này chúng tôi nghiên cứu phiên bản Energy-Constrained DFS / Piecemeal DFS là một biến thể DFS giới hạn năng lượng hoặc tài nguyên, yêu cầu tác nhân sau mỗi hành trình phải quay về điểm xuất phát để "sạc lại" hoặc "nạp lại năng lượng" ứng dụng với hai dạng đồ thị được cho trước.

Từ khóa: *PDFS, Thuật toán tìm kiếm cây, Mức năng lượng rang buộc, Hợp hai cây.*

1. Đặt vấn đề

Thuật toán DFS đã được sử dụng rộng rãi trong chuyên ngành toán rời rạc để áp dụng vào các dạng toán trên lý thuyết đồ thị. Một thuật toán định tuyến dựa trên tìm kiếm theo chiều sâu cho các mạng máy tính được kết nối theo đồ thị hình sao. Thuật toán có thể định tuyến các thông điệp từ một nút nguồn đến đích khi có nhiều liên kết bị lỗi trong mạng [1]. Thuật toán DFS khám phá không gian tìm kiếm của tất cả các lệnh loại trừ, có kích thước n!, trong đó n là số biến trong mạng Bayesian. Để giảm thiểu những vấn đề này, một thuật toán khác được đề xuất cho tìm kiếm theo chiều sâu mở rộng (EDFS). Thuật toán EDFS mới giới thiệu hai kỹ thuật sau đây như là những cải tiến cho thuật toán DFS: (1) một thuật toán duy trì clique động mới chỉ tính toán những clique có chứa một cạnh mới và (2) một quy tắc cắt tỉa mới, được gọi là cắt tỉa clique trục [2]. Trong một đề xuất khác, một thuật toán DFS bộ nhớ ngoài mới cho đồ thị lưới tổng quát. Hiệu suất của thuật toán này tốt hơn so với các thuật toán đã biết khác cho lớp đồ thị này [3]. Một phương pháp được đề xuất với thuật toán Tìm kiếm theo chiều sâu được sửa đổi để tạo ra các trường hợp kiểm thử mong đợi. Bài báo này chứng minh rằng thuật toán DFS đã sửa đổi được áp dụng để tạo trường hợp kiểm thử cung cấp kết quả chính xác, mọi nút được trình bày trên trường hợp kiểm thử, bao gồm bất kỳ điều kiện nào [4].

Một dạng ứng dụng khác là kiểm tra vấn đề về các tác nhân di động (robot) có năng lượng hạn chế khi khám phá một cây. Số lượng các cạnh mà một tác nhân duy nhất có thể đi qua bị giới hạn bởi hạn chế năng lượng. Mục tiêu là giảm quy mô của nhóm tác nhân mà chúng ta sử dụng để cùng nhau khám phá cây. Các tác nhân bắt đầu tại một nút duy nhất, là gốc được chỉ định của cây. Chiều cao của cây được cho là thấp hơn giới hạn năng lượng B của các tác nhân [5]. Một dạng biến thể khác là thuật toán tìm kiếm nhanh theo chiều sâu (DSFS) được đề xuất để cải thiện tốc độ lập kế hoạch của thuật toán Cây ngẫu nhiên khám phá nhanh nhanh. Mối quan hệ bất bình đẳng về chi phí giữa tổ tiên và con cháu của nó đã được suy ra và các tổ tiên đã được lọc theo đó. Đồng thời hệ thống lập kế hoạch đường dẫn dẫn đường hỗ trợ trọng lực dưới nước đã được thiết kế dựa trên thuật toán DSFS, có tính đến

tính phù hợp, an toàn và tính tối ưu tiệm cận của các tuyến đường, theo phân phối tính phù hợp của trọng lực của không gian dẫn đường [6].

Một đề xuất thuật toán có thể liệt kê các tập hợp giả đóng trong các thứ tự không nhất thiết phải mở rộng thứ tự bao gồm bằng cách sử dụng tìm kiếm theo chiều sâu trong một chuỗi các hệ thống đóng [7]. Thuật toán giải quyết vấn đề khám phá bị ràng buộc của một đồ thị chưa biết G = (V, E) từ một nút bắt đầu s cho trước bằng một robot có dây buộc hoặc một robot có bình nhiên liệu có sức chứa hạn chế, trường hợp trước là ràng buộc chặt chẽ hơn. Trong cả hai biến thể của vấn đề, robot chỉ có thể di chuyển dọc theo các cạnh của đồ thị, tức là nó không thể nhảy giữa các đỉnh không liền kề. Trong trường hợp robot có dây buộc, nếu dây buộc (dây thừng) có chiều dài l, thì robot phải duy trì trong khoảng cách l từ nút bắt đầu s. Trong biến thể thứ hai, một bình nhiên liệu có sức chứa hạn chế buộc robot phải quay trở lại s sau khi đi qua C cạnh [8].

Tìm kiếm theo chiều sâu là một kỹ thuật thuật toán tự nhiên để xây dựng một tuyến đường khép kín đi qua mọi đỉnh của một đồ thị. Độ dài của một tuyến đường như vậy bằng, trong một cây có trọng số cạnh, gấp đôi tổng trọng số của tất cả các cạnh của cây và đây là tối ưu tiệm cận đối với mọi chiến lược khám phá. Bài báo này xem xét một biến thể của các chiến lược tìm kiếm như vậy, trong đó độ dài của mỗi tuyến đường bị giới hạn bởi một số nguyên dương B [9]. Điều đáng ngạc nhiên là kết quả của chúng tôi cho thấy tìm kiếm theo chiều sâu hiệu quả đối với việc khám phá cây bị ràng buộc về năng lượng, mặc dù biết rằng điều tương tự không đúng đối với việc khám phá đồ thị tùy ý bị ràng buộc về năng lượng. Nội dung của nghiên cứu này là mở rộng thuật toán piecemeal-DFS cho hai cây cho trước nhằm tối ưu hóa mức năng lượng được sử dụng đồng thời hợp nhất các thành phần cây đã duyệt để tạo thành cây mới.

2. Các nội dung chính

2.1. Thuật toán Piecemeal-DFS (PDFS): Một thuật toán thỏa các điều kiện sau

i) Với j_0=0 thì v_{j0}=v_0=r,

ii) Với B>1 thì tồn tại một phương án duyệt sao cho

$$d(r, v_{j_{i-1}}) + l(v_{j_{i-1}}, v_{j_{i-1}+1}, \dots, v_p) + d(v_p, r) \le B, \quad (1)$$

iii) Tại đỉnh mà R_i dừng thì v_p là giá trị xa nhất, ký hiệu là v_{j_i},

iv) Biểu diễn R_i=P_{i-1}°$(v_{j_{i-1}}, v_{j_{i-1}+1}, \dots, v_p)$°$P_i^R$, với P_{i-1} là đường đi từ r tới $v_{j_{i-1}}$ và P_i^R là đường đi từ v_{j_i} tới r, được gọi là PDFS hay B-exploration.

2.1. Một số tính chất:

Định lý 1: Cho T_1 và T_2 là hai cây rời nhau, mỗi cây có độ sâu không vượt quá B/2. Gọi PDFS($T_1 \cup T_2$) là chiến lược Piecemeal DFS được áp dụng trên cả hai cây với cách duyệt từng cây một và có thể quay lại gốc nhiều lần. R là chiến lược tối ưu chi phí của B-chiến lược khám phá dưới giới hạn năng lượng B, khi đó:

$$PDFS(T_1 \cup T_2) \le 12.|R|, \quad (2)$$

với $PDFS(T_1 \cup T_2) = |PDFS(T_1)| + |PDFS(T_2)|$, và $|R| = |R_1| + |R_2|$, với R_1, R_2 là số tuyến tối thiểu cho mỗi cây.

Chứng minh:

Mỗi cạnh bị duyệt tối đa 6 lần: 2 lần DFS gốc +4 lần dư thừa do chia tour. Tổng chi phí PDFS:

$$PDFS(T_i)|\leq 6\cdot w(T_i) \quad (3)$$

với $w(T_i)$ là tổng trọng số cạnh trong cây T_i.

Theo lý thuyết duyệt cây tối ưu dưới ràng buộc năng lượng, chiến lược tối ưu ít nhất cũng phải bao phủ toàn bộ cạnh, đi + về, nên chi phí tối thiểu gấp đôi tổng trọng số cây, ta có:

$$w(T_i)\leq 2\cdot|R_i| \quad (4)$$

Kết hợp (3) và (4) ta được điều phải chứng minh.

Kết quả: Nếu mỗi cây con T_i (i = 1, 2) có độ sâu ≤ B/2, thì khi duyệt toàn bộ $T_1 \cup T_2$, Piecemeal DFS sẽ không tốn quá 12 lần chi phí tối ưu.

Định lý 2: Với mỗi cây con T_1 và T_2 đều có độ sâu B/2 thì chi phí Piecemeal DFS được xác định như sau:

$$\varepsilon(PDFS(T_1 \cup T_2)) \leq 12.\varepsilon(COPT(T_1 \cup T_2)) , \quad (5)$$

Trong đó, ε(PDFS(·)): tổng chi phí (năng lượng) cần thiết để đi hết cây bằng thuật toán Piecemeal DFS (nhiều lượt, mỗi lượt ≤ B); ε(COPT(·)): chi phí **tối ưu** nhỏ nhất để đi hết cây trong một chuyến, không giới hạn năng lượng.

Chứng minh

Ta gọi $w(T_i)$ là tổng trọng số các cạnh của cây T_i (chiều dài tất cả các cạnh). Chiến lược duyệt DFS đầy đủ sẽ đi qua mỗi cạnh hai lần (đi và quay lại), nên:

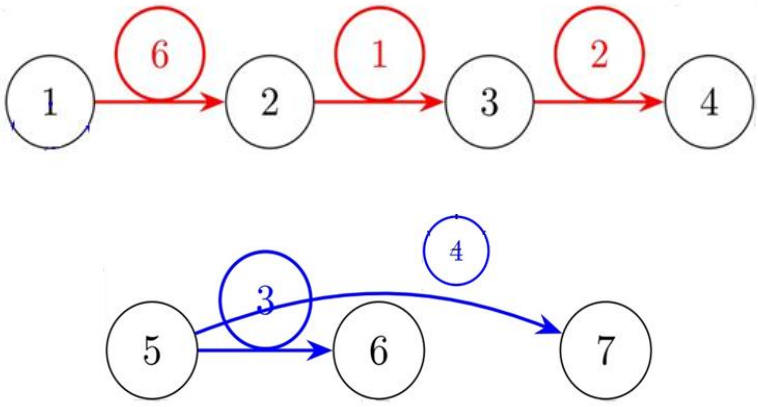
$$\varepsilon(COPT(T_i))=2\cdot w(T_i) \quad (6)$$

Với cây có độ sâu ≤B/2, thì Piecemeal DFS duyệt mỗi cạnh tối đa 6 lần, tức:

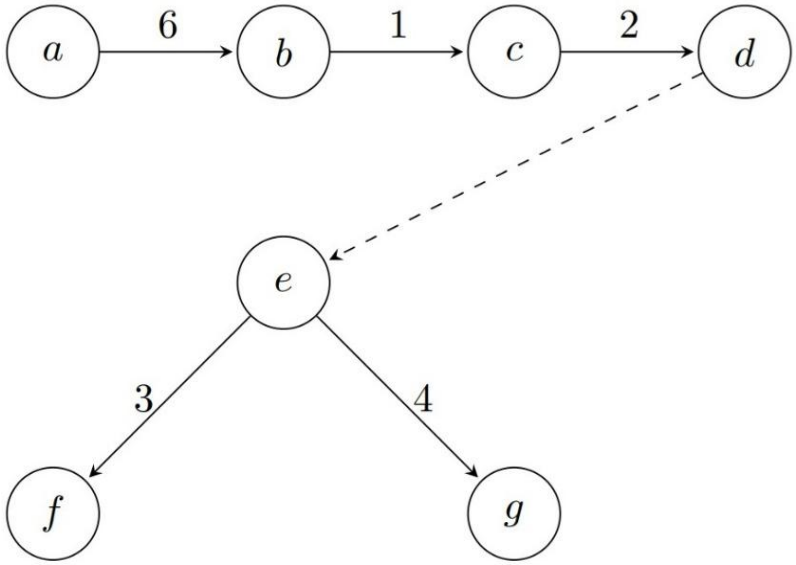
$$\varepsilon(PDFS(T_i))\leq 6\cdot 2\cdot w(T_i)=12\cdot w(T_i) \quad (7)$$

Sử dụng (6) và (7), ta suy ra điều phải chứng minh.

Kết quả của định lý nêu tổng số route tối thiểu khi duyệt riêng 2 cây là ≥ tổng chi phí tối ưu chia B của từng cây. Nội dung chính của phần thuật toán sẽ áp dụng trên hai cây cho trước được minh họa ở Hình 1 và Hình 2.



Hình 1. *Minh họa hai cây cho trước, hai cây rời nhau: không có cạnh nối giữa chúng. Mỗi cây tương ứng xem là T_1 và T_2.*



Hình 2. *Lộ trình của piecemeal-DFS cho hai cây. Cạnh nối giữa c và e là nét đứt, tức là không có liên kết trực tiếp gọi là hai cây rời nhau T_1 và T_2.*

Để mô tả thuật toán, một số ví dụ sẽ được minh họa cụ thể như sau:

Ví dụ 1: Minh hoạ thuật toán duyệt PDFS

Cây 1:	Cây 2:
a | b | c | d	e / \ f g

Giả sử giới hạn năng lượng B=20

Di chuyển trên cây 1:

1. a →b: trọng số 6 → tổng = 6
2. b →c: trọng số 1 → tổng = 7
3. c →d: trọng số 2 → tổng = 9

Chuyển sang cây 2 (còn năng lượng):

4. e →f: trọng số 3 → tổng = 12
5. e →g: trọng số 4 → tổng = 16

Ví dụ 2: Minh họa PDFS bằng cách gộp hai cây rồi duyệt

Cây 1 (gốc là đỉnh 1):

1
/ \
2 3

Cây 2 (gốc là đỉnh 4):

4
/
5

Gộp hai cây lại bằng cạnh (1 - 4) với trọng số 2:

1
/ \ \
2 3 4
\
5

Ma trận kề đầu vào (trước khi thêm cạnh nối):

$$\begin{bmatrix} 0 & 2 & 3 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Quá trình duyệt PIECE_DFS(1):

1. Tại đỉnh 1, duyệt các đỉnh kề:
    - (1 → 2): cost = 2 → năng lượng còn lại = 8
        - Đến đỉnh 2, không còn nhánh con ⇒ quay về 1
    - (1 → 3): cost = 3 → năng lượng còn lại = 5
        - Đến đỉnh 3, không còn nhánh con ⇒ quay về 1
    - (1 → 4): cost = 2 → năng lượng còn lại = 3
        - Đến đỉnh 4:
            - (4 → 5): cost = 1 → năng lượng còn lại = 2
                - Đến đỉnh 5, hết nhánh ⇒ kết thúc

Tổng năng lượng đã tiêu tốn:

- (1 → 2): 2
- (2 → 1): implicit return (không tính vì DFS không quay lại)
- (1 → 3): 3
- (1 → 4): 2
- (4 → 5): 1

Tổng năng lượng tiêu tốn: 2 + 3 + 2 + 1 = 8.

Ví dụ 3: Minh họa PDFS bằng cách duyệt hai cây riêng lẻ xong mới gộp cây

Khi duyệt 2 cây riêng biệt:

- Cây 1:

- Duyệt: 1 → 2 → 3
- Năng lượng tiêu: 2 (1→2) + 3 (1→3) = 5

- Cây 2:
  - Duyệt: 1 → 2 (tương ứng 4→5)
  - Năng lượng tiêu: 1

Tổng năng lượng: 5 + 1 = 6

Kết luận:

- Duyệt riêng 2 cây tiêu tốn ít năng lượng hơn (6 < 8) nhưng tốn thêm lượt.
- Gộp cây thuận tiện nếu muốn khám phá liền mạch, nhưng đôi khi không tối ưu về chi phí (do thêm cạnh kết nối).

Bên cạnh việc trình bày thuật toán cho chiến lược duyệt hai cây, việc so sánh với việc hợp nhất hai cây cho trước rồi áp dụng thuật toán cũng được thực hiện. Bảng 1 mô tả chi tiết hai quá trình thực hiện này.

Bảng 1. *So sánh PDFS cho trường hợp duyệt hai cây.*

Yếu tố	**Duyệt riêng từng cây**	**Gộp hai cây rồi duyệt**
Tổng năng lượng	Tối đa là tổng chi phí 2 PDFS	Có thể tiết kiệm nếu có phần trùng route
Sự linh hoạt khi B nhỏ	Dễ kiểm soát giới hạn năng lượng	Có thể tốn nhiều B chỉ để di chuyển giữa hai cây
Phần trùng lặp route	Có sẵn chặn trên: ≤12·(OPT1+OPT2)	Có thể gom route để tiết kiệm (nếu hai cây gần nhau)
Phân tích lý thuyết	Độ phân giải	Chặn trên vẫn là ≤12·OPT(T) nhưng có thể **lớn hơn** tổng hai cái riêng lẻ nếu nối tốn kém
Triển khai thực tế	Dễ quản lý hơn	Phức tạp nếu không rõ cấu trúc hai cây hoặc liên kết yếu

Mã giả Piecemeal DFS cho hai cây T_1 và T_2

THUẬT TOÁN PiecemealDFS(T, s, B):

Đầu vào:

- T: ma trận kề của cây cần duyệt

- s: đỉnh gốc của cây

- B: năng lượng tối đa cho mỗi lượt

Đầu ra:

- PATH: đường đi các đỉnh theo Piecemeal DFS

Khởi tạo STACK ← rỗng

Đánh dấu s là đã thăm

Thêm s vào PATH

Đưa (s, B) vào STACK

Trong khi STACK không rỗng:

(u, energy) ← lấy đỉnh trên cùng STACK và loại khỏi STACK

Với mỗi đỉnh kề v của u:

Nếu chưa thăm v và energy ≥ T[u][v]:

Đánh dấu v đã thăm

Thêm v vào PATH

Đưa lại (u, energy) vào STACK // Quay về để duyệt tiếp

Đưa (v, energy - T[u][v]) vào STACK

Thoát khỏi vòng lặp (tiếp tục DFS từ v)

Ngược lại nếu không đủ năng lượng:

Đưa lại (s, B) vào STACK // Bắt đầu lượt mới từ gốc

Thoát khỏi vòng lặp

Trả về PATH

3. Độ phức tạp của thuật toán:

Giả sử tác nhân duyệt T_1 rồi T_2 hoặc ngược lại, với mỗi cây, độ phức tạp PDFS là

$$O(n_i \cdot \frac{D_i}{B})$$

trong đó, $n_i = |V_i|$ là số đỉnh của cây T_i, D_i là đường kính (khoảng cách xa nhất giữa hai đỉnh) của cây T_i, B là năng lượng tối đa của mỗi hành trình. Do đó, tổng độ phức tạp có được là

$$O(n_1 \cdot \frac{D_1}{B} + n_2 \cdot \frac{D_2}{B})$$

Ngoài ra, tổng chi phí hợp nhất hai cây là $O(h \cdot f(n))$, với h là số hành trình và f(n) là chi phí mỗi lần hợp nhất. Do đó, tổng độ phức tạp ước lượng toàn bộ thuật toán là

$$O(n_1 \cdot \frac{D_1}{B} + n_2 \cdot \frac{D_2}{B} + h \cdot f(n))$$

Trong đó, h là số hành trình cần thiết, f(n) là chi phí hợp nhất mỗi lần $O(1)$ hoặc $O(n)$.

Cây gộp có kích thước gần như $n=n_1+n_2$ đối với việc gộp hai cây trước rồi duyệt:

- Đường kính D lớn hơn hoặc bằng $\max(D_1, D_2)$, do cây nối thêm cạnh
- Chỉ duyệt 1 lần Piecemeal DFS trên cây gộp:

$$O\left(n \cdot \frac{D}{B}\right) = O((n_1 + n_2) \cdot \frac{D}{B})$$

- Không cần nhiều lần hợp nhất, nên bỏ qua h·f(n) hoặc coi là rất nhỏ.

4. Kết quả nghiên cứu và thảo luận

Bài báo đã thể hiện dạng mở rộng của thuật toán PDFS cho trường hợp duyệt hai cây cho trước với mức năng lượng cố định nhằm tối ưu hóa năng lượng và duyệt hai cây hoàn toàn. Bài toán có nhiều ứng dụng trong việc tìm đường đi cho robot với mức năng lượng được cung cấp từ vị trí đầu tiên sau mỗi lần trở lại của robot. Kết quả của quá trình duyệt hai cây cho trước tốt hơn khi nhận dạng hai cây riêng biệt và sau đó hợp nhất hai cây. Tuy nhiên, việc hợp nhất hai cây trước quá trình duyệt sẽ tốn nhiều năng lượng hơn. Kết quả này cho biết việc xác định đường đi ở các thành phần liên thông riêng lẻ sẽ đạt mức tối ưu năng lượng hơn so với việc duyệt một hệ thống chung (Bảng 1). Piecemeal DFS khi mở rộng cho phần ($n \geq 2$) cây vẫn giữ được tính hiệu quả trong môi trường hạn chế tài nguyên, đồng thời mở ra nhiều chiến lược tối ưu hóa về năng lượng, thời gian và phân bổ tài nguyên.

5. Kết luận

Nghiên cứu đã th ể hiện thành công việc sử dụng thuật toán PDFS mở rộng cho trường hợp duyệt hai cây cho trước. Nghiên cứu cũng đã xét trường hợp duyệt hai cây nhẹ sẽ tiết kiệm năng lượng hơn so với duyệt một cây nặng bằng cách hợp nhất hai cây cho trước [9]. Kết quả thể hiện này là tiền đề quan trọng cho việc tổng quát hóa cho việc sử dụng thuật toán để duyệt n thành phần ($n \geq 2$) liên thông. Piecemeal DFS hoàn toàn có thể sử dụng cho nhiều hơn 2 cây vì PDFS hoạt động cục bộ, các cây là độc lập, mỗi cây đều có thể áp dụng PDFS riêng, việc phối hợp có thể tối ưu hóa thông qua thứ tự duyệt, hợp nhất, hoặc phân công tác nhân.

Tài liệu tham khảo:

[1]. Sur, S., & Srimani, P. K. (1994). A depth-first search routing algorithm for star graphs and its performance evaluation. Mathematical and Computer Modelling, 19(9), 35–52. https://doi.org/10.1016/0895-7177(94)90039-6

[2]. Li, C., & Ueno, M. (2017). An extended depth-first search algorithm for optimal triangulation of Bayesian networks. International Journal of Approximate Reasoning, 80, 294–312. https://doi.org/10.1016/j.ijar.2016.09.012

[3]. Her, J.-H., & Ramakrishna, R. S. (2007). An external-memory depth-first search algorithm for general grid graphs. Theoretical Computer Science, 374(1–3), 170–180. https://doi.org/10.1016/j.tcs.2006.12.022

[4]. Meiliana, Septian, I., Alianto, R. S., Daniel, & Gaol, F. L. (2017). Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm. Procedia Computer Science, 116, 629–637. https://doi.org/10.1016/j.procs.2017.10.029

[5]. Das, S., Dereniowski, D., & Karousatou, C. (2018). Collaborative Exploration of Trees by Energy Constrained Mobile Robots. Theory of Computing Systems, 62(5), 1223–1240. https://doi.org/10.1007/s00224-017-9816-3.

[6]. Zhou, X., Zheng, W., Li, Z., Wu, P., & Sun, Y. (2024). Improving path planning efficiency for underwater gravity-aided navigation based on a new depth sorting fast search algorithm. Defence Technology, 32, 285–296. https://doi.org/10.1016/j.dt.2023.04.012

[7]. Bazin, A. (2018). A depth-first search algorithm for computing pseudo-closed sets. Discrete Applied Mathematics, 249, 28–35. https://doi.org/10.1016/j.dam.2018.03.030

[8]. Duncan, C. A., Kobourov, S. G., & Kumar, V. S. A. (2006). Optimal constrained graph exploration. ACM Transactions on Algorithms, 2(3), 380–402. https://doi.org/10.1145/1159892.1159897

[9]. Das, S., Dereniowski, D., & Uznański, P. (2024). Energy Constrained Depth First Search. Algorithmica, 86(12), 3759–3782. https://doi.org/10.1007/s00453-024-01275-8